



### Packet-less traffic analysis using Wireshark

#### Luca Deri <deri@ntop.org>, @lucaderi Samuele Sabella <sabella@ntop.org>

ntop





- Luca is the founder of the ntop project that develops open source network traffic monitoring applications. All code is available at https://github.com/ntop
- Samuele is an undergraduate student at the Computer Science Department of the University of Pisa. His interests include networking and machine learning. He's an ntop team member.
- ntop is a community: http://t.me/ntop\_community
- We are part of the Intel Innovator program.





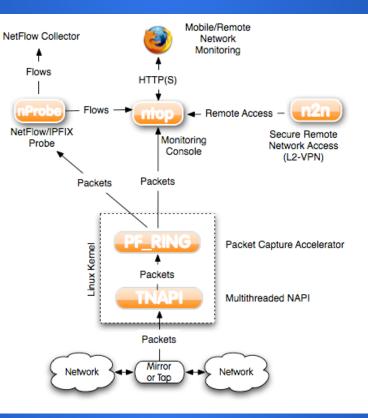
- Since the early days network monitoring has focused on packets. Indeed Wireshark is a <u>packet</u> <u>analyser</u>.
- Packets are used to deliver user data across applications.
- Users/applications have no packet visibility as they are a "low level" concept used at network layer.

# (We Used to Say) Packets Never Lie



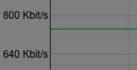
- Packet analysis provide useful information for understanding:
  - Network traffic issues.
  - Network usage not compliant with network policies (note: firewalls cannot help here).
  - Non-optimal performance.
  - Potential security flaws.
  - Ongoing (latent) attacks.
  - Data breach.

# ntop Ecosystem (2009): Packets



## Interface: eno1 About to extract traffic from 13:10:10 to 13:14:59

Extract pcap 
Packets



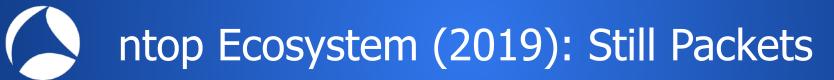
Traffic -

938.62 Kbit/s

480 Kbit/s

320 Kbit/s

n <mark>o</mark> o
erc 60
50 40
30





0

Х



Nothing in general, but they offer an external viewpoint

- From which we have to reconstruct what is really happening from the application/user standpoint.
- Good for monitoring network traffic from outside of a systems on a passive way (no agent installation required).
- Packets are low level and need to be "interpreted" in order to understand what happens at a higher layer: TCP zerowindow, fragments, and packet retransmissions are invisible to applications and users that instead think in terms of perceived network performance and transmission errors.

## What's Wrong with Packets? [2/2]



- Packets resemble synthetic information and lack of metadata that help understanding insights on the machine
- Data encryption is a challenging for DPI techniques and Wireshark, making more complicated packet payload to be dissected and decoded.
- Network administrators need to monitor packets fragmentation, flow reconstruction, packet loss/ retransmissions... metrics that would be already available inside a system but that instead are measured with packets.

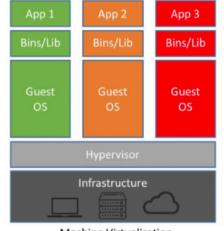
#### Provide an additional layer of isolation over processes. Allow each service to use its own dependencies.



# What about Containers?

App 1App 2App 3Bins/LibBins/LibBins/LibContainer EngineOperating SystemInfrastructure

Make services portable across host platforms.

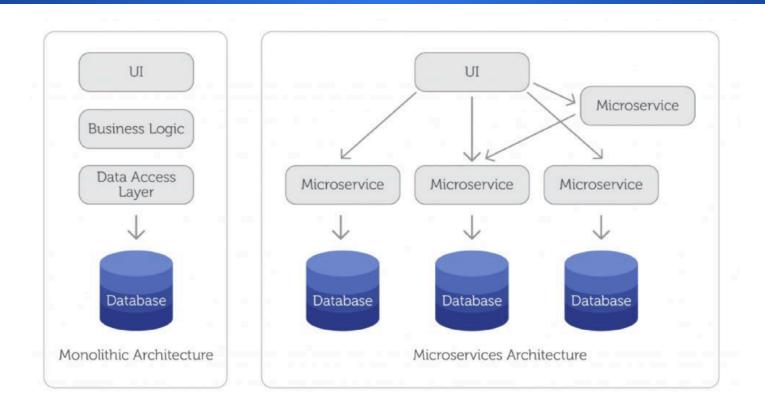


Machine Virtualization



# From Monolith to Microservices [1/3]







- Code of each microservice is stored in an isolated container, runs its own memory space, and functions independently.
- Clearly organised architecture. Decoupled units have their specific jobs, can be reconfigured without affecting the entire application.
- Deployments don't require downtime.



- If a microservice crashes, the rest of the system keeps going.
- Each microservice can be scaled individually according to its needs.
- Services can use different tech stacks (developers are free to code in any language, and HR are happy to hire programmers that do not necessarily have to code in the same programming language).

# Networking and Namespaces



- In Linux network interfaces and routing tables/ entries are shared across the entire OS.
- Sometimes (containers need that) it is necessary to define different and separate instances of network interfaces and routing tables that operate independent of each other
- Linux implements this using namespaces.

# Playing with Namespaces [1/4]



#### Create a new namespace

# ip netns add wireshark

# ip netns list cni-53bd89ab-d120-4015-0fc8-f5cb5ed45413 cni-f4c00b32-2487-8e9c-3f60-e5d425aaa1d7 (id: 2) cni-0ce982f1-b6ac-2035-9ee0-a9cd8eb8d9d6 (id: 1) cni-920496f6-b76f-a6e0-145f-4fa315134140 (id: 0) wireshark Playing with Namespaces [2/4]



### • Create a new veth interface peer

# ip link add veth0 type veth peer name veth1

#### # ip link list

- 1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
- 2: enp0s5: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc fq\_codel state UP mode DEFAULT group default qlen 1000

link/ether 00:1c:42:85:41:62 brd ff:ff:ff:ff:ff:ff

... ... ... ...

8: **veth1@veth0**: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000

link/ether 5e:cb:a9:10:50:e9 brd ff:ff:ff:ff:ff:ff

9: **veth0@veth1**: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000

```
link/ether ca:b4:d6:da:7c:b0 brd ff:ff:ff:ff:ff
```

Playing with Namespaces [3/4]



#### • Bind a veth to a namespace

#### # ip link set veth1 netns wireshark

#### # ip netns exec wireshark ip link list

- 1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
- 8: veth1@if9: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000 link/ether 5e:cb:a9:10:50:e9 brd ff:ff:ff:ff:ff:ff link-netnsid 0

### • Configure an IP address

# ip netns exec wireshark ifconfig veth1 192.168.10.1/24 up

# ip netns exec wireshark ifconfig veth1

veth1: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
 inet 192.168.10.1 netmask 255.255.255.0 broadcast 192.168.10.255
 ether 5e:cb:a9:10:50:e9 txqueuelen 1000 (Ethernet)
 RX packets 0 bytes 0 (0.0 B)
 RX errors 0 dropped 0 overruns 0 frame 0
 TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

#sf19eu • Palacio Estoril Hotel, Estoril, Portugal • Nov 4 - 8

## Playing with Namespaces [4/4]



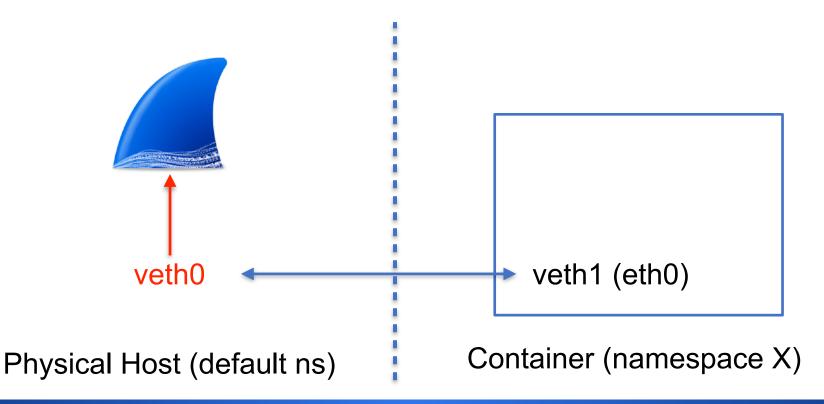
# Adding a veth to a physical network via a bridge

# brctl addif cbr0 veth0

# brctl show
bridge name bridge id
cbr0 8000.6a870bb63548

STP enabled interfaces no veth0 veth5a9abc1c veth884b5ab2 vethc6499b6e vethd3260294





### What's Wrong with Packets on Containerised Environments? [1/2]



- Virtualisation techniques reduce visibility when monitoring network traffic as network manager are blind with respect to what happens inside the systems.
- Each container has a virtual ethernet interface so commands such as "tcpdump -i veth40297a6" won't help as devops think in terms of container name, pod and namespace rather than veth.
- Intra-container traffic stays inside the system without hitting the wire, thus monitoring traffic from/to the host won't help.

### What's Wrong with Packets on Containerised Environments? [2/2]



- Containers are not VMs which have a "long-range" time cycle
- Environments like Kubernetes are extremely dynamic.
- It's hard to associate an IP address to a service because addresses have become ephemeral.
- System introspection can help us correlating the network traffic with the continuously moving parts of our infrastructure.



• Even on a container-centric sites we still need to:

- Monitor the infrastructure where containers are deployed: SNMP, NetFlow/IPFIX, and packets/Wireshark.
- Enable system introspection also to (legacy) noncontainerised systems so the whole infrastructure is monitored seamlessly.
- This means that we need to <u>enable Wireshark to</u> <u>be used on those containerised environments</u>.



Challenges using Wireshark with Containers [1/2]



- Intra-container traffic will never hit the wire: sniffing on eth0 won't help.
- It is not intuitive to bind a veth interface to a container name/pod in order to sniff the container traffic:
  - Each containerised environment has its own tools and naming (kubernetes != docker, Linux ns != Kubernets namespaces).
  - Interfaces appear/disappear as container are created/ deleted.



Challenges using Wireshark with Containers [2/2]



- As a container pool (pod) often offers a service by load-balancing the traffic across multiple containers, it is not intuitive to follow a packet journey when passing across NAT and balancing.
- This problem will be discussed later in this presentation.

# From Challenges to Solutions



- Enhance network visibility leveraging on system introspection for <u>adding new</u> <u>metadata to network packets</u>, this in order to ease troubleshooting.
- Handle virtualisation as first citizen and don't be blind (yes we want to observer containers interaction).
- Complete our monitoring journey and...
  - System Events: processes, users, containers.
  - $\circ$  Flows
  - Packets
- Bind metadata captured from system events at the application layer (e.g. tcp\_connect invocation) to the network traffic for enabling continuous drill down: system events uncorrelated with network traffic are basically useless.



### Do we still need DPI?



- In this world DPI (Deep Packet Inspection) has marginal importance since we have information on the process that generated the network event
  - User, group
  - process, absolute path, pid,
  - container id, pod, namespace
- If we are able to know that an application generated a network event and then we are able to bind that information to the network traffic then DPI makes less sense.

## **Design Goals**



• Extend Wireshark to take into account system events in order to provide some context (process, user, PID...) to the captured traffic.



- Hide Wireshark the complexity of containerised environments and let network administrators focus on packet analysis without them being container experts.
- <u>IMPORTANT: We don't want to replace packet capture with</u> <u>events but rather complement captured traffic with</u> <u>additional information.</u>

## Early Experiments: Sysdig



- Provides a way to observe the system at the kernel system call level.
- ntop has been an early sysdig adopter adding in 2014 sysdig events support in ntop tools.
- Despite all our efforts, this activity has NOT been a success for many reasons:
  - Too much CPU load (in average +10-20% CPU load)

Svsdia

- requires a new kernel module that sometimes is not what sysadmins like as it might invalidate distro support.
- Containers were not so popular in 2014, and many people did not consider system visibility so important at that time.





### How Sysdig Works



- As sysdig focuses on system calls for tracking a TCP connections we need to:
  - Discard all nonTCP related events (sockets are used for other activities on Linux such as Unix sockets)
  - Track socket() and remember the socketId to process/thread.
  - Track connect() and accept() and remember the TCP peers/ports.
  - Collect packets and bind each of them to a flow (i.e. this is packet capture again, using sysdig instead of libpcap).

This explains the CPU load, complexity...

## Using Sysdig [1/2]



\$ curl https://www.ntop.org

#### # sysdig -pc evt.type=connect or evt.type=bind

25395 16:56:35.648903745 0 host (host) curl (26431:26431) > connect fd=3(<u>) 25396 16:56:35.648914011 0 host (host) curl (26431:26431) < connect res=-2(ENOENT) tuple=0->ffff9458c020ec00 /var/run/nscd/socket 25401 16:56:35.648922620 0 host (host) curl (26431:26431) > connect fd=3(<u>) 25402 16:56:35.648924967 0 host (host) curl (26431:26431) < connect res=-2(ENOENT) tuple=0->ffff9458c020ec00 /var/run/nscd/socket 25537 16:56:35.649282362 0 host (host) curl (26431:26431) > connect fd=3(<4>) 25538 16:56:35.649289899 0 host (host) curl (26431:26431) < connect res=0 tuple=131.114.21.11:42026->131.114.21.6:53 25699 16:56:35.650580211 0 host (host) curl (26431:26431) > bind fd=3(<n>) 25700 16:56:35.650582767 0 host (host) curl (26431:26431) < bind res=0 addr=NULL 25721 16:56:35.650631926 0 host (host) curl (26431:26431) > connect fd=3(<6>) 25724 16:56:35.650642514 0 host (host) curl (26431:26431) < connect res=0 tuple=2a00:d40:1:3:131.114.21:11:41764->2a03:b0c0:2:d0::360:4001:443 25727 16:56:35.650645184 0 host (host) curl (26431:26431) > connect fd=3(<6>2a00:d40:1:3:131.114.21:41764->2a03:b0c0:2:d0::360:4001:443) 25728 16:56:35.650645950 0 host (host) curl (26431:26431) < connect res=0 tuple=0.0.0.0:0->0.0.0.0:0 25729 16:56:35.650646881 0 host (host) curl (26431:26431) > connect fd=3(<4u>0.0.0.0:0->0.0.0.0:0) 25732 16:56:35.650650936 0 host (host) curl (26431:26431) < connect res=0 tuple=::ffff:131.114.21.11:45555->::f87c;a283;c1a3;ffff;443 25810 16:56:35.652983176 5 host (host) curl (26430:26430) > connect fd=3(<6>) 25811 16:56:35.653036637 5 host (host) curl (26430:26430) < connect res=-115(EINPROGRESS) tuple=2a00:d40:1:3:131.114.21:11:60894->2a03:b0c0:2:d0::360:4001:443

## Using Sysdig [2/2]



\$ top   grep -E	D sy	sdig ebp		1					
PID USER	PR	NI	VIRT	RES	SHR S	%CPU	ы́МЕМ	TIME+	COMMAND
25197 root	20	0	351116	16404	11552 S	0,7	0,2	0:01.49	sysdig
25133 root	20	0	159668	122128	48100 S	0,3	1,5	0:03.74	ebpflowexport
25197 root	20	0	351116	16404	11552 R	11,6	0,2	0:01.84	sysdig
25133 root	20	0	159668	122128	48100 S	3,3	1,5	0:03.84	ebpflowexport
25197 root	20	0	351116	16404	11552 S	26,7	0,2	0:02.65	sysdig
25133 root	20	0	159668	122128	48100 S	0,3	1,5	0:03.85	ebpflowexport
25133 root	20	0	159668	122128	48100 S	0,7	1,5	0:03.87	ebpflowexport
						and the second second			

Load matters in particular on the cloud

### Towards eBPF



- 1992: Steve McCane and Van Jacobson introduced a VM model packets filtering. This version of BPF is now known as classic BPF (cBPF)
- 1997: cBPF was introduced in Linux in kernel 2.1.75, as a technology for inkernel packet filtering
- 2013: eBPF, created by Alexei Starovoitov, extended what bpf virtual machine could do. The VM is now able to intercept other kind of events and take several action other than filtering (https://lkml.org/lkml/2013/12/2/1066)





### Welcome eBPF



- eBPF is great news for Wireshark as:
- It gives the ability to avoid sending everything to user-space but perform in kernel computations and send metrics to user-space.
- We can track more than system calls (i.e. be notified when there is a transmission on a TCP connection without analyzing packets).
- It is part of modern Linux systems (i.e. no kernel module needed).

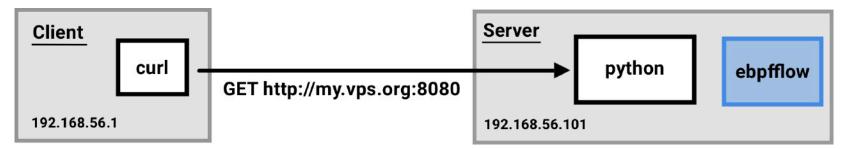
# libebpflow: eBPF for System Visibility



- Our aim has been to create an open-source library that offers a simple way to interact with eBPF network events in a transparent way.
  - Reliable and trustworthy information on the status of the system when events take place.
  - Low overhead event-based monitoring
  - Information on users, network statistics, containers and processes
  - Go and C/C++ support
- https://github.com/ntop/libebpfflow (GNU LGPL)







• We host a service on port 8080

user@Server:~/\$ python -m SimpleHTTPServer 8080

• We use curl to http-GET using the local port 1234 user@Client::~/\$ curl --local-port 1234 http://my.vps.org:8080

## libebpflow: client-server [2/2]

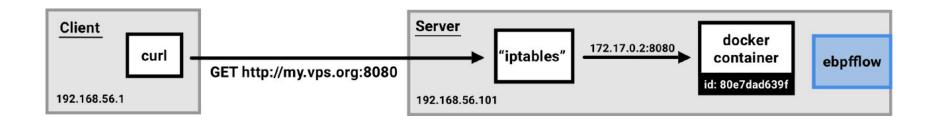


#### 📶 🔳 🖉 💿 🚞 🖺 🖄 🙆 🔍 🖛 🌧 警 🖉 🖢 📃 📃 🍳 Q. Q. 🏢

o. Time Sour 1 0.0000 N/A					
1 0 0000 N/A				Info	
	N/A	eBPF		TCP_Close	
2 0.3504 N/A	N/A	eBPF		UDP_Send	
3 0.5358 N/A	N/A	eBPF		UDP_Send	
4 2.5868 N/A	N/A	eBPF	384	UDP_Send	
5 6.0015 N/A	N/A	eBPF		TCP_Close	
6 8.0121 N/A	N/A	eBPF		TCP_Close	
7 9.5331 N/A	N/A	eBPF		TCP_Accept	
8 9.5336 N/A	N/A	eBPF	384	TCP_Close	
Kernel time ( Interface nam Event time (s Event time (u Event IP prot Event vpe: T IPv4 src addr IPv4 dst addr Event protoco Event source Event destina	ec): 3019582880 isec): 16902 :: enp088 ec): 1573048007 iec): 575310 iec): version: 4 if received P_Accept ess: 192.168.56.1 iss: 192.168.56.101 :: TCP port: 51427 ion port: 8080				







- We Run detached container which serves https on port 80 user@Server:~/\$ docker run --rm -it -p 4443:8080 sabellas/cowserve
- We use curl to https-GET using the local port 1234 user@Client:~/\$ curl --local-port 1234 http://my.vps.org:8080



### 🚄 🔳 🖉 🐵 🚞 🖺 🕅 🖄 🙆 🔍 🖛 🌧 警 🖌 👤 📃 🔍 Q. Q. 🎛

#### Apply a display filter ... <Ctrl-/>

No.	-	Time	Source	Destination	Protocol	Length	Info
	1	0.0000	N/A	N/A	eBPF	384	TCP Accept
	2	0.0038	N/A	N/A	eBPF	384	TCP_Close
	3	3.1182	N/A	N/A	eBPF	384	TCP_Accept
	4	3.1393	N/A	N/A	eBPF	384	TCP_Close
	5	3.4550	N/A	N/A	eBPF	384	UDP Send
	6	3.4550	N/A	N/A	eBPF	384	UDP Send

#### Frame 3: 384 bytes on wire (3072 bits), 384 bytes captured (3072 bits) on interface 0 Null/Loopback

- Data (380 bytes)
- eBPFFlow Protocol
  - Kernel time (sec): 3388946263 Kernel time (usec): 17091 Interface name: eth0 Event time (sec): 1573051946

Event IP protocol version: 4

Event type: TCP\_Accept IPv4 src address: 192.168.56.1 IPv4 dst address: 172.17.0.2 Event protocol: TCP Event source port: 52382 Event destination port: 8080

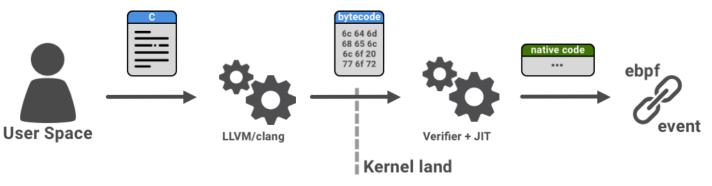
Event retransmissions: 0 Event Process PID: 16576 Event Process TID: 16576 Event Process UID: 0 Event Process GID: 0 Event Process Task: node Event Father PID: 16553 Event Father TID: 0

#### and the second second second second

Event Father GID: 0 Event Father Task: containerd-shim Event Container Id: 80e7dad639f858734cbaf53bf507d29d819db97e6f896c6b2c149df00413ed92 Event Container name: web-server

## Under the Hood

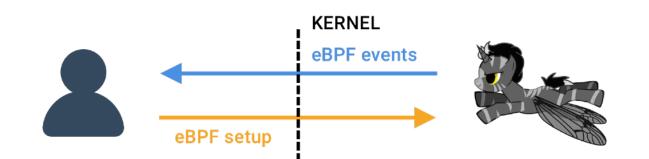




- The user writes a program in C
- The program is translated in eBPF instructions (LLVM/clang)
- A verifier check if the eBPF program is safe (e.g. no loops, only known external function allowed)
- A just in time compiler translate the bytecode directly into a target architecture: x86, ARM, MIPS, etc.
- The program is attached to the target kernel event such that whenever the event is triggered the program is executed







```
// Attaching probes ----- //
if (userarg_eoutput && userarg_tcp) {
    // IPv4
    AttachWrapper(&ebpf_kernel, "tcp_v4_connect",
                                                   "trace connect entry",
                                                                              BPF PROBE ENTRY);
    AttachWrapper(&ebpf_kernel, "tcp_v4_connect",
                                                                              BPF PROBE RETURN);
                                                   "trace_connect_v4_return",
    // IPv6
    AttachWrapper(&ebpf_kernel, "tcp_v6_connect",
                                                                              BPF PROBE ENTRY);
                                                  "trace connect entry".
    AttachWrapper(&ebpf_kernel, "tcp_v6_connect",
                                                   "trace connect v6 return",
                                                                              BPF PROBE RETURN);
}
```

# libebpfflow Overview [2/2]



typedef enum { eTCP ACPT = 100, eTCPCONN = 101, eTCP CONN FAIL = 500, eUDP RECV = 210, eUDP SEND = 211, eTCP RETR = 200, eTCP CLOSE = 300, event type; struct taskInfo { u32 pid; /\* Process Id \*/ u32 tid: /\* Thread Id \*/ u32 uid; /\* User Id \*/ u32 gid; /\* Group Id \*/ char task[COMMAND LEN], \*full task path; }; // separate data structs for ipv4 and ipv6 struct ipv4 addr t { u64 saddr; u64 daddr: **};** struct ipv6 addr t { unsigned \_\_int128 saddr; unsigned int128 daddr; **};** 

typedef struct {
 ktime\_t ktime;
 char ifname[IFNAMSIZ];
 struct timeval event\_time;
 u\_int8\_t ip\_version, sent\_packet;
 u16 etype;

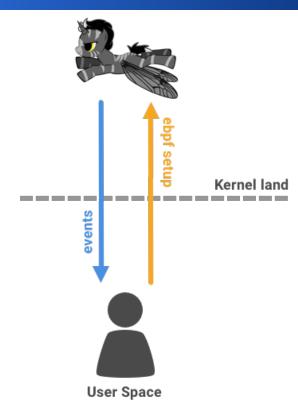
union {
 struct ipv4\_addr\_t v4;
 struct ipv6\_addr\_t v6;
} addr;

u8 proto; u16 sport, dport; u32 latency\_usec; u16 retransmissions;

struct taskInfo proc, father; char container\_id[CONTAINER\_ID\_LEN];

struct {
 char \*name;
} docker;

struct {
 char \*name;
 char \*pod;
 char \*ns;
 kube;
} eBPFevent;



# Collecting Information: Processes

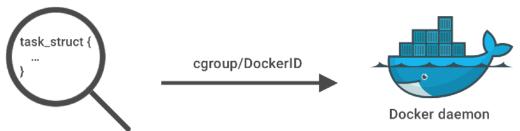


- In linux every task has associated a struct (i.e. task\_struct) that can be retrieved by invoking the function bpf\_get\_current\_task provided by eBPF. By navigating through the kernel structures it can be gathered:
  - uid, gid, pid, tid, process name and executable path
  - cgroups associated with the task.
- Connection details instead are read from the socket structure. They include: source and destination ip/port, bytes send and received, protocol used.

# Collecting Information: Containers

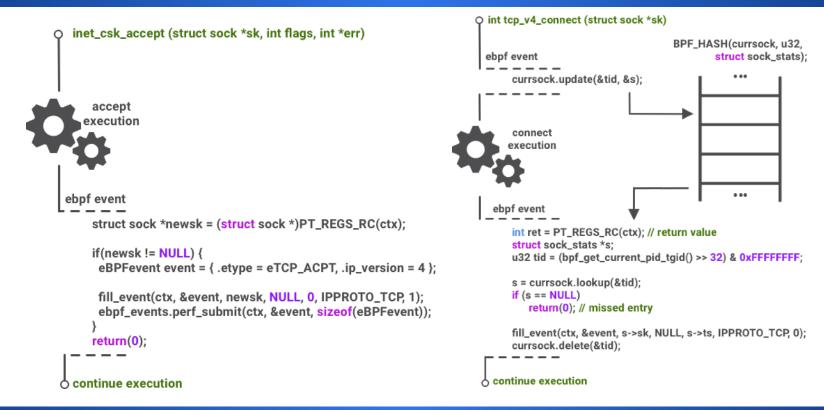


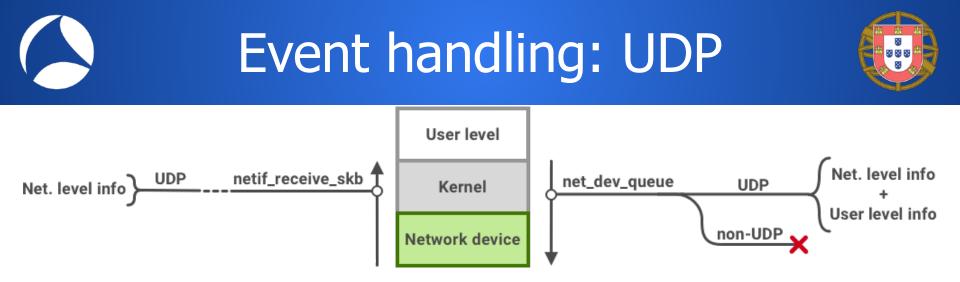
- Container can be found in proc/cgroup, however retrieving information from there is a too slow operation.
- Because containers are processes, we can navigate through kernel data structures and read information from inside the kernel where the container identifier can be collected.
- Further interaction with the container runtimes (e.g. containerd or dockerd) in use is required to collect detailed information



# Event handling: TCP







 In order to capture UDP events we attach eBPF code to net\_dev\_queue (process send buffer for network) and netif\_receive\_skb (process receive buffer from network) tracepoints and discard non UDP events.





## Wireshark/libebpflow Integration





- We have developed an open source Wireshark extension that enabled network traffic monitoring by leveraging on network events. It can be used:
  - by installing a Wireshark plugin
  - from the CLI
  - or... by running a container
- The tool capture all network events within a system providing information both at
  - network level: addresses and ports
  - user level: users and processes

## For the Impatient...



📮 ntop / libebpfflow		O Unwatch →5★ Star70% Fork8
<>Code (!) Issues 0	1 Pull requests 1 🔲 Projects 0 💷 Wiki 🕕 Security	🔟 Insights 🛛 🔅 Settings
	fflow / wireshark /	Create new file Upload files Find file History
Iucaderi Added kubernet	s pod name to the interface name	Latest commit 3b61264 2 days ago
Makefile	Interface display fixes	16 days ago
README.md	Network interfaces are now listed using kubectl	24 days ago
🖹 ebpf.lua	Added eBPF parsing on ethernet trailer	17 days ago
ebpfdump.c	Added kubernets pod name to the interface name	2 days ago
■ pcapio.c	Network interfaces are now listed using kubectl	24 days ago
🖹 pcapio.h	Network interfaces are now listed using kubectl	24 days ago

### https://github.com/ntop/libebpfflow/tree/master/wireshark

# Option 1: Events Only Monitoring



• •	•			debpf_events.pcapn	g			
	📕 🔬 🛞		🗙 🙆 🔍 🖛 🖷	) 🚈 🖣 👱	🔳 🔍 २, २,	<b>I</b>		
App	oly a display filter	<೫/>				Expressi	on +	
۱o.	Time	Source	<ul> <li>Destination</li> </ul>	Protocol	Info			
	21 0.783175	N/A	N/A	eBPF	UDP_Send			
	22 1.076825	N/A	N/A	eBPF	TCP_Connect			
	23 1.076951		N/A	eBPF	TCP_Accept			
	24 1 077662	M 7 A	N/A	ARDE	TCB Close			
Nui ⊨ Dai ▼ eBi	<pre>ll/Loopback ta (364 bytes) ta (364 bytes) PFFlow Protocy Kernel time ( Event source Event source Event addret Event destime Event latency Event retrans Event Process Event Process Event Process Event Process Event Process Event Process Event Father Event Father Event Father Event Father Event Father Event Father Event Father Event Father Event Father Event Contain 00 00 07 c3 00 00 00 00</pre>	ol sec): 732572: usec): 1643 we: cbr0 ec): 1570968 sec): 801290 ocol version on: Sent CP_Connect ess: 10.1.1.1 l: TCP port: 35238 tidon port: 11 ( usec): 147 missions: 0 PID: 8926 i TID: 8927 i TID: 8927 i TID: 8927 i TID: 8273 uID: 0 Task: kubele PID: 1 TID: 0 GID: 0 Task: system er Id: es 2a aa 2b 00 00 00 00 9a 99 00 00	<pre>485 2 4 1 (10.1.1.1) 8 (10.1.1.0) 00054 et d 60 06 00 063 62 72 00 00 08 00 a5 13 a3</pre>	30 k.	•cbre		Te: Default	vents (no packets)

### Option 2: Events and Packets



<ul> <li>Frame 207: 222 bytes on wire (1776 bits), 2</li> <li>Ethernet II, Src: 82:aa:2a:6b:ed:a3 (82:aa:</li> <li>Internet Protocol Version 4, Src: 10.1.1.1</li> </ul>	2a:6b:ed:a3), Dst: 52:47:27:a5:5b:41 (52:47:27:a5:5b:41)
▶ Transmission Control Protocol, Src Port: 34	1400 (34400), Dst Port: 10054 (10054), Seq: 3477950944, Ack: 2160430723, Len: 118
<ul> <li>Hypertext Transfer Protocol</li> <li>GET /metrics HTTP/1.1\r\n</li> </ul>	
Host: 10.1.1.6:10054\r\n User-Agent: kube-probe/1.15\r\n Accept-Encoding: gzip\r\n	
Connection: close\r\n	
\r\n	Legacy Wireshark (packets
[Full request URI: http://10.1.1.6:10054	<u>/metrics]</u> Leyacy Vileshaik (packets
[HTTP request 1/1]	
Event Process PID: 20059	
Event Process TID: 3939	
Event Process UID: 0	(Glued) Network Event
Event Process GID: 0	
Event Process Task: kube-pro	
Event Container Id: kube-proxy	





- As explained before, system events are not received on a network interface but they over a kernel-to-userspace queue.
- As Wireshark is unable to handle non networkinterfaces, the best solution for bringing events into it was to develop an extcap tool.

	Mer	ging Wireshark with eBPF [2/2]
	•••	X The Wireshark Network Analyzer
		Go Capture Analyze Statistics Telephony Wireless Tools Help
		) 💼 🖹 🕅 🗳 👄 🔿 🖄 💽 📃 🔍 Q, Q, X
	Apply a display f	filter <ctrl-></ctrl-> Expression   +
		Welcome to Wireshark Development Build
		Capture
Extcap		using this filter: 📕 Enter a capture filter 🔹 5 interface(s) shown, 15 hidden 👻
	<u>.</u>	DisplayPort AUX channel monitor capture: dpauxmon
eBPF		BepF interface: ebpf
		Bandom packet generator: randpkt     Extcap interface: xport: sdjournal
Module		ebpfdump     No capture filter     ve capture: udpdump



## What is Extcap?



- "The extcap interface is a versatile plugin interface that allows external binaries to act as capture interfaces directly in wireshark".
- In essence it defines a set of command line conventions to interface external tools to send wireshark captured packets (even on nonnetwork interfaces) via a named pipe.

https://www.wireshark.org/docs/man-pages/extcap.html

# ebpfdump [1/2]



\$ ebpfdump
Wireshark extcap eBPF plugin by ntop

Supported command line options: --extcap-interfaces --extcap-version --extcap-dlts --extcap-interface <name> --extcap-config --capture --fifo <name> --debug --name <name> --custom-name <name> --help

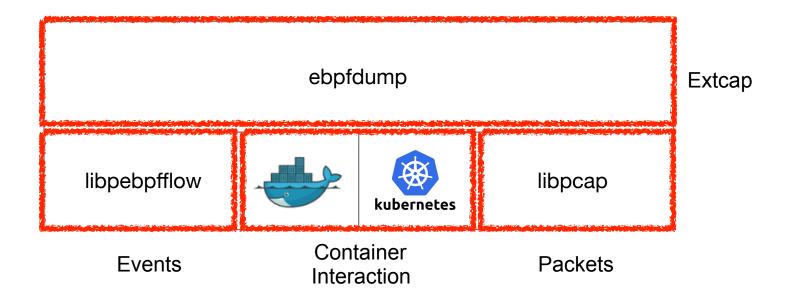
Wireshark         Authors         Folders         Plugins         Keyboard Shortcuts         Acknowledgments         License           Filter by path	0 0		X About Wireshark		
Name         Location         Typical Files           "File" dialogs         capture files           Temp         /tmp         untitled capture files           Personal configuration         /usr/share/wireshark         dfilters, preferences, ethers,           Global configuration         /usr/share/wireshark         dfilters, preferences, ethers,           Program         /usr/bin         /tmp           Program         /usr/bin         program files           Personal Plugins         /home/deri/.local/lib/wireshark/plugins/3.1         binary plugins           Global Plugins         /usr/lib/x86.64-linux/wireshark/plugins/3.1         binary plugins	Wireshark Authors	Folders Plug	ins Keyboard Shortcuts	Acknowledgments	License
"File" dialogs capture files untitled capture files untitled capture files Personal configuration / usr/share/wireshark/ dfilters, preferences, ethers, dfilters, preferences, manuf, ethers, ipprate thers, ipprate there, ipprate there there, ipprate there, ipprate there there, ipprate there, ipprate there there, ipprate there there, ipprate there there, ipprate there there there, ipprate there there, ipprate there t	Filter by path				
Extcap path /usr/lib/x86 64-linux-qnu/wireshark/extcap Extcap Plugins search path MaxMind DB path /usr/share/GeoIP MaxMind DB database search path MaxMind DB path /usr/share/GeoIP MaxMind DB database search path MaxMind DB path /usr/share/GeoIP MaxMind DB database search path MB/PIB path SMI MIB/PIB search path	Name "File" dialogs Temp Personal configuration Global configuration System Program Personal Plugins Global Plugins Global Lua Plugins <u>Stcap path</u> MaxMind DB path MaxMind DB path	/tmp /home/deri/.confi /usr/share/wiresh /etc /usr/lib/x86 64-lir /home/deri/.local /usr/lib/x86 64-lir /usr/lib/x86 64-lir /usr/share/GeolP /var/lib/GeolP	ark lib/wireshark/plugins/3.1 rux/wireshark/plugins/3. lib/wireshark/plugins ux-gnu/wireshark/plugins	capture files untitled capture files dfilters, preferences, dfilters, preferences, ethers, ipxnets program files binary plugins 1 binary plugins 1 ua scripts lua scripts <b>Extap Plugins search</b> MaxMind DB databass MaxMind DB databass	manuf, 1 path e search path e search path e search path

# ebpfdump [2/2]



- \$ ebpfdump --extcap-config --extcap-interface ebpf
- arg {number=0}{call=--ifname}{display=Interface Name}{type=selector}{tooltip=Network Interface from which
  packets will be captured}
- value {arg=0}{value=ebpfevents}{display=eBPF Events}
- value {arg=0}{value=ebpfzmqevents}{display=eBPF Remote Events (ZMQ)}
- value {arg=0}{value=veth73d654ec}{display=Pod kube-dns-6bfbdd666c-5jbmx, Namespace kube-system}
- value {arg=0}{value=veth02c998da}{display=Pod monitoring-influxdb-grafana-v4-78777c64c8-k8c26, Namespace
  kube-system}
- value {arg=0}{value=cbr0}{display=cbr0}
- value {arg=0}{value=veth3b09c8fd}{display=veth3b09c8fd}
- value {arg=0}{value=flannel.1}{display=flannel.1}
- value {arg=0}{value=enp0s5}{display=enp0s5}
- value {arg=0}{value=veth1e9ce659}{display=veth1e9ce659}
- value {arg=0}{value=lo}{display=lo}
- value {arg=0}{value=docker0}{display=docker0}





# ebpfdump Operating Modes [1/5]



- (a) eBPF events: only eBPF events are returned (no packets).
  - Events are dumped as they are received and delivered to Wireshark in pcap format.
  - A lua dissector companion file decodes the received events and show them in human friendly mode.

# ebpfdump Operating Modes [2/5]



Apply a display filter _ <%/> <ul> <li></li></ul>	•	•			ebpf_events	pcapng	
Time         Source         Destination         Protocol         Infe           19         0.7831_         N/A         N/A         eBPF         UDP_Send           20         0.7831_         N/A         N/A         eBPF         UDP_Send           21         0.7831_         N/A         N/A         eBPF         UDP_Send           Event         IP protocol version: 4         Event direction: Received         Event source port: 10050         Event source port: 35238           Event source port: 35238         Event festination port: 10054         Event festination port: 10054         Event festination port: 10054           Event Process TDD: 10061         Event Process TDD: 10061         Event festination port: 10054         Event festination port: 10054           Event Process TDD: 10061         Event Process TDD: 10061         Event Process TDD: 10061         Event Process TDD: 10061           Event Father TDD: 0         Event Father TDD: 0         Event Father TDD: 0         Event Father TDD: 0           Event Father Task: containerd-shim         Event Container ID: 14225a72cbadf8756d8061ebb05be7f20d33be73772535a602174a4c708ff95           090         00 00 00 00 00 00 00 00 00 00 00 00 00	(	📕 🙍 🛞		🚺 🤇 🖛 🛋	) 🖀 🚡	👱 📃 📕 ભ્ ભ્ ભ્ 🏢	
19 0.7831       N/A       N/A       eBPF       UDP_Send         20 0.7831       N/A       N/A       eBPF       UDP_Send         21 0.785       UDC_Scott       EVent       rotes:       Location:         21 0.787       N/A       eBPF       UDP_Send       LDP_Send         Event fathery:       cdathost:       (127.0.0.1)       LDP       LDP         Event fathery:       socialist:       (127.0.0.1)       LDP       LDP       LDP         Event retransmissions:	Ap	ply a display filter	<9€/>				Expression +
20 0.7831       N/A       N/A       eBPF       UDP_Send         21 0.7831       N/A       N/A       eBPF       UDP_Send         Event IP protocol version: 4         Event direction: Received       Event type: TCP_Accept         IPv4 str address: localhost (127.0.0.1)       IPv4 str address: localhost (127.0.0.1)         Event source port: 35238       Event destination port: 10054         Event vorces pr1: 9440       Event Process TD: 10001         Event Process TD: 10001       Event Process GD: 65534         Event Father TD: 0       Event Father TD: 0         Event Father TD: 0       Event Father GD: 0         Event Father GD: 0       Event Stask: containerd-shim         Event Container Id: 14225af2cbadf8756d8061ebb05be7f20d33be73772535a602174a4c708ff95         000 00 00 00 00 00 00 00 00 00 00 00 00	s.	Time	Source	<ul> <li>Destination</li> </ul>	Protocol	Info	
21 0.7831		19 0.7831	N/A	N/A	eBPF	UDP_Send	
Event IP protocol version: 4 Event UP protocol version: 4 Event direction: Received Event type: TCP_ACcept IPv4 src address: localhost (127.0.0.1) IPv4 dst address: localhost (127.0.0.1) Event source port: 35238 Event destination port: 10054 Event source port: 35238 Event destination port: 10054 Event Process FID: 9440 Event Process FID: 9440 Event Process GID: 65534 Event Process GID: 65534 Event Process GID: 65534 Event Father TID: 0 Event Father Tid: 642C5af2Cbad061ebb05be7f20d33be73772535a602174a4c708ff95 <b>1009</b> 00 00 00 00 00 00 00 00 00 00 00 63 6f 6e 74 <b>1009</b> 00 00 00 00 00 00 00 00 00 00 00 63 6f 6e 74 <b>1000</b> Event Father Tid: 642C5af2Cbad061ebb05be7f20d33be73772535a602174a4c708ff95 <b>1000</b> 00 00 00 00 00 00 00 00 00 00 00 00		20 0.7831	N/A	N/A	eBPF	UDP_Send	
Event direction: Received Event type: TCP_Accept IPv4 src address: localhost (127.0.0.1) IPv4 dst address: localhost (127.0.0.1) Event protocol: TCP Event source port: 35238 Event destination port: 10054 Event generations: 0 Event fratersmissions: 0 Event Process FID: 9440 Event Process FID: 9440 Event Process GDI: 65534 Event Process GDI: 65534 Event Process GDI: 65534 Event Father FID: 0 Event Father FID: 0 Event Father GDI: 0		21 0.7831	N/A	N/A	eBPF	UDP_Send	
Event direction: Received Event type: TCP_Accept IPv4 src address: localhost (127.0.0.1) IPv4 dst address: localhost (127.0.0.1) Event protocol: TCP Event source port: 35238 Event destination port: 10054 Event generations: 0 Event fratersmissions: 0 Event Process FID: 9440 Event Process FID: 9440 Event Process GDI: 65534 Event Process GDI: 65534 Event Process GDI: 65534 Event Father FID: 0 Event Father FID: 0 Event Father GDI: 0							
Event type: TCP_Accept IPv4 src address: localhost (127.0.0.1) IPv4 dst address: localhost (127.0.0.1) Event protocol: TCP Event protocol: TCP Event source port: 35238 Event destination port: 10054 Event destination port: 10054 Event Process FDI: 9440 Event Process TD: 10001 Event Process GDI: 65534 Event Process GDI: 65534 Event Process GDI: 65534 Event Process TD: 9963 Event Father TDI: 0 Event Father TDI: 0 Event Father GDI: 0 Event Father GDI: 0 Event Father IDI: 0 Event Father IDI: 0 Event Father IDI: 0 Event Father IDI: 0 Event Father JS: containerd-shim Event Container Id: f4c2C5af2cbadf8756d8061ebb05be7f20d3be73772535a602174a4c708ff95 ************************************		Event IP pro	otocol versio	n: 4			
IPv4 src address: localhost (127.0.0.1) IPv4 dst address: localhost (127.0.0.1) IPv4 dst address: localhost (127.0.0.1) Event protocol: TCP Event source port: 35238 Event destination port: 10054 Event tetransmissions: 0 Event retransmissions: 0 Event Process PID: 9440 Event Process GDI: 65534 Event Process GDI: 65534 Event Process GDI: 65534 Event Process GDI: 65534 Event Father PID: 9963 Event Father TDI: 0 Event Father GDI: 0 Event Father JTS: 60 00 00 00 00 00 00 00 00 00 00 00 00		Event direct	tion: Receive	d			
IPv4 dst address: localhost (127.0.0.1) Event protocol: TCP Event source port: 35238 Event destination port: 10054 Event tetransmissions: 0 Event process TD: 9440 Event Process TD: 10001 Event Process TD: 65534 Event Process GD: 65534 Event Process GD: 65534 Event Process TD: 963 Event Father TD: 0 Event Father TD: 0 Event Father GD: 0 Event Father GD: 0 Event Father GD: 0 Event Father Task: containerd-shim Event Container Id: 14225af2cbad18756d8061ebb05be7f20d3be73772535a602174a4c708ff95 Went Father GD: 0 Event Father GD							
Event protocol: TCP Event source port: 35238 Event destination port: 10054 Event destination port: 10054 Event latency (usec): 0 Event Process TID: 9440 Event Process TID: 10001 Event Process TID: 10001 Event Process GID: 65534 Event Process GID: 65534 Event Process GID: 65534 Event Process GID: 65534 Event Process Task: sidecar Event Father PID: 9963 Event Father TID: 0 Event Father GID: 0 Event Father GID: 0 Event Father GID: 0 Event Father III: 0 Event Container I f42c25af2cbadf875668061ebb05be7f20d3be73772535a602174a4c708ff95 ************************************							
Event       source port:       35238         Event       destination port:       10054         Event       tatency (usec):       0         Event       retransmissions:       0         Event       process PID:       9440         Event       process PID:       9440         Event       process PID:       10001         Event       process GD:       65534         Event       Process GD:       65534         Event       Pacess Task:       sidecar         Event       Father TD:       0         Event       Father GD:       0         Event       Father GD:       0         Event       Father Task:       containerd-shim         Event       Father Task:       containerd-shim         Event       616 96 66 57 26 42 d7 38 86 96 d0 09 92 e2 a9 95       ainerd-s him····         000       00 00 66 33 26 33 56 16 63 22 63 26 51 73 33 73 73 72       ber/12/d3 3be/3772535a6/2174a4/708/f95         000 62 65 76 63 23 36 64 38 33 62 65 37 73 33 73 73 72       be/72/d3 3be/3772         000 63 53 56 66 30 32 21 31 73 46 11 46 33 73 30 85 53 66621 74a4/c708       f95				ost (127.0.0.1)			
Event destination port: 10054 Event latency (usec): 0 Event retransmissions: 0 Event Process TD: 9440 Event Process TD: 10001 Event Process TD: 65534 Event Process TD: 65534 Event Process TD: 9963 Event Father TD: 0 Event Father TD: 0 Event Father GD: 0 Event Father GD: 0 Event Father Task: containerd-shim Event Father Task: containerd-shim Event Father Task: containerd-shim Event 6 6 66 57 26 4 2d 73 68 69 6d 00 90 2e ab 95 61 69 6e 65 72 64 2d 73 68 69 6d 00 90 2e ab 95 61 69 6e 65 72 64 2d 73 68 69 6d 00 90 2e ab 95 61 69 6e 65 72 64 2d 73 68 69 6d 00 90 2e ab 95 61 69 6e 65 72 64 2d 73 68 69 6d 00 90 2e ab 95 61 69 6e 65 72 64 2d 73 68 69 6d 00 90 2e ab 95 61 69 6e 65 72 64 2d 73 73 68 69 6d 00 90 2e ab 95 61 69 6e 65 72 64 2d 73 73 73 73 73 72 be7720d3 3be73772 62 64 66 65 33 73 55 66 43 33 33 62 65 37 73 37 37 32 be7720d3 3be73772 60 80 80 80 80 80 80 80 80 80 80 80 80 80							
Event latency (usc): 0         Event retransmissions: 0         Event Process JD: 9440         Event Process TD: 10001         Event Process TD: 165534         Event Process GD: 65534         Event Process Task: sidecar         Event Father PID: 9963         Event Father TD: 0         Event Father GD: 0         Event Father GD: 0         Event Father GD: 0         Event Father Tdf 14 f42265472cbadf875668061ebb05be7f20033be73772535a602174a4C708ff95         00       00 00 00 00 00 00 00 00 00 00 00 00 02 ca b 95         01 69 6e 65 72 64 2d 73 68 69 6d 00 99 2e a b 95         02 64 66 38 37 35 36 64 38 30 36 31 65 62 62 30 35         02 00 64 66 38 37 35 36 64 33 33 62 65 37 33 37 37 32         02 00 80 00 00 00 00 00 00 00 00 00 00 00							
Event retransmissions: 0         Event Process PID: 9440         Event Process PID: 10001         Event Process UD: 65534         Event Process GD: 65534         Event Process GD: 65534         Event Process GD: 65534         Event Process GD: 65534         Event Father PID: 9963         Event Father TD: 0         Event Father GD: 0         Event Father GD: 0         Event Father Id: 14225a2badf8756d8061ebb05be7f20d33be73772535a602174a4c708ff95         000 00 00 00 00 00 00 00 00 00 00 00 00				10034			
Event Process PID: 9440         Event Process TID: 10001         Event Process TID: 65534         Event Process GID: 65534         Event Process Task: sidecar         Event Process Task: sidecar         Event Father PID: 9963         Event Father TID: 0         Event Father GID: 0         Event Father TID: 0         Event Father Task: containerd-shim         Event Container Id: 14225312Cbad1875648061ebb05be7120d33be73772535a602174a4c7081f95         000       00 00 00 00 00 00 00 00 00 00 00 02 2e ab 95         ainerd-s him · · · · · · · · · · · · · · · · · · ·							
Event         Process         GS534           Event         Process         GS534           Event         Process         GS534           Event         Process         GS534           Event         Father PID:         9963           Event         Father TID:         0           Event         Father TID:         0           Event         Father GID:         0           Event         Father GID:         0           Event         Father Task: containerd-shim							
Event Process GID: 65534         Event Process Task: sidecar         Event Father PID: 9963         Event Father TID: 0         Event Father TID: 0         Event Father TID: 0         Event Father TID: 0         Event Father TID: 10         Event Father Task: containerd-shim         Event Container Id: f4c2c5af2cbadf8756d8061ebb05be7f20d33be73772535a602174a4c708ff95         1090       00 00 00 00 00 00 00 00 00 00 00 63 6f 6e 74         0100 b3 55 00 00 66 34 63 32 63 35 61 66 22 63 62 61 40.1.4.4.4.4.4.4.4.4.4.4.4.4.4.4.4.4.4.4		Event Proces	ss TID: 10001				
Event Process Task: sidecar         Event Father PID: 9963         Event Father TD: 0         Event Father Task: containerd-shim         Event Container Id: f42c5af2cbadf8756d8061ebb05be7f20d33be73772535a602174a4c708ff95         1099       00 00 00 00 00 00 00 00 00 00 00 00 63 6f 6e 74         0109       00 00 00 00 00 00 00 00 00 00 00 00 00		Event Proces	ss UID: 65534				
Event Father PID: 9963         Event Father TID: 0         Event Father TID: 0         Event Father UD: 0         Event Father GID: 0         Event Father Task: containerd-shim         Event Container Id: f4c2c5af2cbadf8756d8061ebb05be7f20d33be73772535a602174a4c708ff95         0090 00 00 00 00 00 00 00 00 00 00 00 05 6 f 6 74         0100 00 00 00 00 00 00 00 00 00 00 00 00							
Event Father TID: 0         Event Father UID: 0         Event Father UID: 0         Event Father Task: containerd-shim         Event Father Task: containerd-shim         Event Container Id: f4c2c5af2cbadf8756d8061ebb05be7f20d33be73772535a602174a4c708ff95         0000 00 00 00 00 00 00 00 00 00 00 06 36 6f 6e 74         0100 00 00 00 00 00 00 00 00 00 00 06 36 6f 6e 74         0100 00 00 00 00 00 00 00 00 00 00 06 36 6f 6e 74         0100 00 00 00 00 00 00 00 00 00 00 00 00				car			
Event Father UID: 0           Event Father GD: 0           Event Father Task: containerd-shim           Event Container Id: 1422531220adf8756d8061ebb05be7f20d33be73772535a602174a42708ff95           090         00 00 00 00 00 00 00 00 00 00 03 6f 6e 74           010 00 00 00 00 00 00 00 00 00 00 00 03 6f 6e 74           010 00 00 00 00 00 00 00 00 00 00 00 03 6f 6e 74           010 00 00 00 00 00 00 00 00 00 00 00 00							
Event Father GID: 0           Event Father Task: containerd-shim           Event Container Id: f4c2Csaf2cbad8756d8061ebb05be7f20d33be73772535a602174a4c708ff95           0090 00 00 00 00 00 00 00 00 00 00 06 63 6f 6e 74           01910 00 00 00 00 00 00 00 00 00 00 63 6f 6e 74           01920 00 00 00 00 00 00 00 00 00 00 00 63 6f 6e 74           01930 01 00 66 34 63 32 63 35 61 66 32 63 62 61           0100 02 65 37 66 32 30 64 38 30 63 11 65 62 62 30 35           0100 02 66 63 33 35 61 36 03 22 31 7 34 61 37 34 61 16 62 62 30 35           0100 02 35 33 56 13 60 32 31 7 34 61 34 63 37 33 7 32           0101 02 35 50 00 00 00 00 00 00 00 00 00 00 00 00							
Event Father Task: containerd-shim           Event Container Id: f4c2c5af2cbadf8756d8061ebb05be7f20d33be73772535a602174a4c708ff95           OPU 00 00 00 00 00 00 00 00 03 6f 6e 74           opu 00 00 00 00 00 00 00 00 03 6f 6e 74           Image: Container Id: f4c2c5af2cbadf8756d8061ebb05be7f20d33be73772535a602174a4c708ff95           OPU 00 00 00 00 00 00 00 00 66 6f 6e 74           Image: Container Id: f4c2c5af2cbadf8756d8061ebb05be7f20d33be73772535a602174a4c708ff95           OPU 00 00 00 00 00 00 00 00 00 00 00 00 00							
Event         Container         Id:         f4c2c5af2cbadf8756d8061ebb05be7f20d33be73772535a602174a4c708ff95           1090         00				inerd-shim			
080       61       69       66       72       64       21       73       68       69       60       09       22       ab       95       ainerd-s       him         010       b3       55       00       66       463       32       63       35       61       63       22       ab       95       ainerd-s       him         020       64       66       38       73       35       66       38       36       31       65       62       61       ·0f422       C5af2cba         020       64       66       32       35       64       33       33       62       65       62       31       display       display       bit       ·1       f422       C5af2cba       0       display       display       bit       ·1       bit       ·1       bit       display					lebb05be7f	20d33be73772535a602174a4c708ff	95
080       61       69       66       72       64       21       73       68       69       60       09       22       ab       95       ainerd-s       him         010       b3       55       00       66       463       32       63       35       61       63       22       ab       95       ainerd-s       him         020       64       66       38       73       35       66       38       36       31       65       62       61       ·0f422       C5af2cba         020       64       66       32       35       64       33       33       62       65       62       31       display       display       bit       ·1       f422       C5af2cba       0       display       display       bit       ·1       bit       ·1       bit       display	00	00 00 00 00	00 00 00 00 0	0 00 00 00 00 63	6f 6e 74		
1000       b3       55       00       00       63       46       33       26       33       56       16       32       63       35       61       66       32       63       35       61       66       32       63       35       61       66       32       63       35       61       66       32       63       35       61       66       32       63       35       61       66       32       35       dta       fta       fta </td <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>							
0000       62       65       37       66       32       30       64       33       33       62       65       37       33       37       37       32       be7f20d3       3be73772         0000       35       33       35       61       36       30       23       13       37       461       34       63       37       30       38       535a621       74a4c708         0100       06       06       00<	Øb						
3000         35         33         35         61         36         30         22         31         37         34         61         34         63         37         30         38         535a6021         74a4c708           1010         66         66         35         35         00							
1010       66       65       35       36       00       00       00       00       00       00       00       100							
0100 00 00 00 00 00 00 00 00 00 00 00 00						53586021 /484C/08	
2110 00 00 00 00 00 00 00 00 00 00 00 00							
8120 80 00 80 80 80 80 80 80 80 80 80 80 80							
	120	00 00 00 00	00 00 00 00 00	0 00 00 00 00 00	00 00 00	••••••	
Z Event Container Id (shofflow container id) 128 bytes Profile: Defa Profile: Defa	_						

# ebpfdump Operating Modes [3/5]



					debpf_events.pcap	ng	
		🦾 🔳 🙋	•	ه 🗣 ک	• 🛎 Ŧ 👱		
		Apply a displa	/ filter <೫/>				Expression +
		No. Time	Source	<ul> <li>Destination</li> </ul>	Protocol	Info	
		19 0.7	31 N/A	N/A	eBPF	UDP_Send	
		20 0.7	331 N/A	N/A	eBPF	UDP_Send	
		21 0.7	331 N/A	N/A	eBPF	UDP_Send	
					0		
				(2944 bits), 360	8 bytes capture	d (2944 bits) on interface 0	
	<u> </u>	Null/Loop					
<b>C</b> vont		▼ Data (364	bytes) bc3ac2b6b0600006574	692000000000000000	000000000000000000000000000000000000000		
Event	- o Sandhining	[Length		002000000000000000000000000000000000000	aaaaaaaaaaaaaaaa		
		v eBPFFlow					
			time (sec): 7327426	19			
		Kernel	time (usec): 1643				
			ce name: eth0				
			ime (sec): 15709684	85			
			ime (usec): 891429				
			P protocol version: irection: Received	4			
			ype: TCP_Accept				
			c address: localhos	t (127.0.0.1)			
			t address: localhos				
		Event p	rotocol: TCP				
			ource port: 35238				
			estination port: 10	054			
			atency (usec): 0				
			etransmissions: 0				
			07 e3 db c3 ac 2b	6b 06 00 00 65		····+ k···eth0	
			00 00 00 00 00 00 00 00 00 25 9a 0d 00	00 00 00 00 a5 00 00 00 00 04		•••••••] •\$*••••••••d•	
			00 00 25 98 00 00 00 00 7f 00 00 01			··············	
			00 00 00 00 00 00	00 00 00 00 00			
			01 01 06 00 a6 89	46 27 00 00 00		····· F!·····	
			00 00 e0 24 00 00 00 00 73 60 64 65	11 27 00 00 fe 63 61 72 00 00		··\$·· ·'····	
			00 00 73 69 64 65 00 00 c0 f2 88 95			····· ·U···&··	
			00 00 00 00 00 00			····· cont	
		😑 🎽 Data (d	ata.data), 364 bytes			<ul> <li>Packets: 38 · Displayed: 38 (100.)</li> </ul>	0%) • Profile: Default

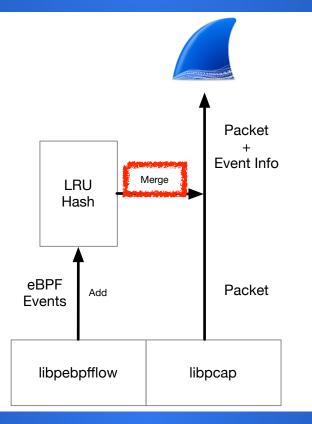
ebpfdump Operating Modes [4/5]



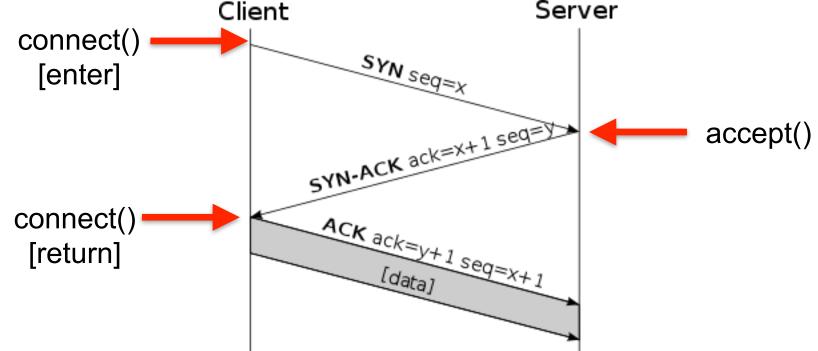
- (b) Packets + eBPF events.
  - Events are received and stored on a LRU hash table that will be used to match packets.
  - Received packets are matched against the LRU hash table and in case of a match, the packet is extended to add event information

# ebpfdump Operating Modes [5/5]









Merging Events with Packets: Timing [1/3]



#### SYN Apply a display filter ... < %/> - -Expression No. Time Destination Protocol Info Source 1 0.000000 PcsCompu\_cc:1d:d3 RealtekU\_12:3... ARP Who has 10.0.2.2? Tell 10.0.2.15 RealtekU\_12:35:02 2 0.000376 PcsCompu cc:1... ARP 10.0.2.2 is at 52:54:00:12:35:02 3 1.002030 10.0.2.15 156.99.224.35... 35676 → http(80) [SYN] Seq=592764910 Win=642 TCP Event 4 1.008520 00:00:00\_00:00:00 00:00:00\_00:0... eBPF TCP\_Connect 5 1.126399 156.99.224.35.bc.g... 10.0.2.15 TCP http(80) → 35676 [SYN, ACK] Seg=1076864001 A 6 1.126516 10.0.2.15 156,99,224,35... TCP 35676 → http(80) [ACK] Seg=592764911 Ack=107 No Merge ▶ Frame 3: 74 bytes on wire (592 bits). 74 bytes captured (592 bits) on interface 0

#### Ethernet II, Src: PcsCompu\_cc:1d:d3 (08:00:27:cc:1d:d3), Dst: RealtekU\_12:35:02 (52:54:00:12:35:02)

Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 156.99.224.35.bc.googleusercontent.com (35.224.99.156)

retransmission.pcapng

👱 📃 📃

Θ. Θ. 🔍 🎹

Transmission Control Protocol, Src Port: 35676 (35676), Dst Port: http (80), Seq: 592764910, Len: 0

### #sf19eu • Palacio Estoril Hotel, Estoril, Portugal • Nov 4 - 8



Merging Events with Packets: Timing [2/3]

📄 🗋 🕅 🏹

۲

(too early)

 $\begin{array}{c|c} \bullet & \bullet & \bullet & \bullet \\ \hline & \bullet & \bullet \\ \hline & \bullet & \bullet \\ \hline & \bullet & \bullet & \bullet \\ \hline & \bullet \\ \hline & \bullet & \bullet \\ \hline & \bullet & \bullet \\ \hline & \bullet \\ \hline & \bullet & \bullet \\ \hline \hline & \bullet \\ \hline \hline & \bullet \\ \hline & \bullet \\$ 

Destination

10.0.2.15

RealtekU 12:3...

PcsCompu cc:1...

00:00:00\_00:0...

156.99.224.35... TCP

156.99.224.35... TCP

Data Merge
(Event + Pkt)

Event

No.

Time

1 0.000000

2 0.000376

3 1.002030

4 1.008520

5 1.126399

6 1.126516

Source

10.0.2.15

10.0.2.15

PcsCompu cc:1d:d3

RealtekU 12:35:02

00:00:00\_00:00:00

156.99.224.35.bc.g...

►	Frame 5: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
►	Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_cc:1d:d3 (08:00:27:cc:1d:d3)
►	Internet Protocol Version 4, Src: 156.99.224.35.bc.googleusercontent.com (35.224.99.156), Dst: 10.0.2.15 (10.0.2.15)
►	Transmission Control Protocol, Src Port: http (80), Dst Port: 35676 (35676), Seq: 1076864001, Ack: 592764911, Len: 0
Ŧ	eBPFFlow Protocol
	Event Process PID: 511
	Event Process TID: 511
	Event Process UID: 0
	Event Process GID: 0
	Event Process Task: NetworkM
	Event Container Id:
ъI	ntop

Protocol

ARP

ARP

eBPF

TCP

Info

TCP\_Connect

### Merging Events with Packets: Timing [3/3]



- -

Who has 10.0.2.2? Tell 10.0.2.15

10.0.2.2 is at 52:54:00:12:35:02

35676 → http(80) [SYN] Seq=592764910 Win=642

http(80) → 35676 [SYN, ACK] Seq=1076864001 A

35676 → http(80) [ACK] Seg=592764911 Ack=107

Expression.

Merging Events with Packets: Naming [1/3]



## • Mapping ContainerIds with Host Interfaces

root@ntop-ubuntu:/home/deri/libebpfflow/utils# ./docker_show_veth.sh	eBPF Events
veth containerId	Container xenodochial_rosalind
	docker0
<pre>vethd38ebdb xenodochial_rosalind</pre>	cbr0
root@ntop-ubuntu:/home/deri/libebpfflow/utils# ifconfig vethd38ebdb	vethd38ebdb
<pre>vethd38ebdb: flags=4163<up,broadcast,running,multicast> mtu 1500</up,broadcast,running,multicast></pre>	flannel.1
<pre>inet6 fe80::9803:15ff:fe41:5b47 prefixlen 64 scopeid 0x20<li (ethernet)<="" 0="" 9a:03:15:41:5b:47="" ether="" pre="" txgueuelen=""></li></pre>	veth215c2ad2
RX packets 65 bytes 5844 (5.8 KB)	enp0s5
RX errors 0 dropped 0 overruns 0 frame 0	veth67e78b7a
TX packets 127 bytes 11706 (11.7 KB) TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0	veth3db12b7c
	veth24b4614d
	lo

Merging Events with Packets: Naming [2/3]



### Container Interface (no vethXXX, won't help)

1570482464.998115 [eth0][Rcvd][IPv4/TCP][pid/tid: 17330/17330 [/usr/bin/python2.7], uid/gid: 0/0][father pid/tid: 17158/0 [/bin/bash], uid/gid: 0/0][addr: 192.168.1.202:54235 <-> 172.17.0.2:80][ACCEPT] [containerID: 79ba73e1213768da608fca002c6b2f5b0c994ce3c4cf62acf1805ebef293b418][docker\_name: xenodochial\_rosalind]

No.	. Time	Source	Destination	Protocol	Length Info
	3 0.000048	172.17.0.2	192.168.1.202	TCP	112 80 → 54235 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=146
	4 0.000282	192.168.1.202	172.17.0.2	TCP	104 54235 → 80 [ACK] Seq=1 Ack=1 Win=131744 Len=0 TSval=69267
	5 0.000750	192.168.1.202	172.17.0.2	HTTP	186 GET / HTTP/1.1
	6 0.000769	172.17.0.2	192.168.1.202	TCP	104 80 → 54235 [ACK] Seq=1 Ack=83 Win=29056 Len=0 TSval=31558
	7 0.001367	172.17.0.2	192.168.1.202	TCP	121 80 → 54235 [PSH, ACK] Seq=1 Ack=83 Win=29056 Len=17 TSval
	8 0.001479	192.168.1.202	172.17.0.2	TCP	104 54235 → 80 [ACK] Seq=83 Ack=18 Win=131744 Len=0 TSval=692
	9 0.001516	172.17.0.2	192.168.1.202	TCP	141 80 → 54235 [PSH, ACK] Seq=18 Ack=83 Win=29056 Len=37 TSva
	10 0 001000	100 100 1 000	170 17 0 0	TCD	104 FADDE DO FACKI CHE OD AND FE WAR 101710 Les O TOURI COD
1000	and the second second second				
					n interface /tmp/wireshark_extcap_ebpf_20191007230738_PQ2vjp, id 0
	Ethernet II, Src:	02:42:ac:11:00:02 (	02:42:ac:11:00:02), Ds	st: 02:42:9d	:03:10:a8 (02:42:9d:03:10:a8)
•	Internet Protocol	Version 4, Src: 172	.17.0.2, Dst: 192.168.	1.202	
	Transmission Contr	ol Protocol, Src Po	rt: 80, Dst Port: 5423	35, Seq: 0,	Ack: 1, Len: 0
•	eBPFFlow Protocol		-		

Merging Events with Packets: Naming [3/3]



- Merging Via Container Name
  - Container Name (Docker)

root 11334 58.6 5.2 232640 106200 pts/1 S 23:16 0:03 /usr/ lib/x86\_64-linux-gnu/wireshark/extcap/ebpfdump --capture --extcapinterface ebpf --fifo /tmp/wireshark\_extcap\_ebpf\_20191007231612\_01Q40m -ifname vethd38ebdb@xenodochial\_rosalind

### • Pod (Kubernetes)

/usr/lib/x86\_64-linux-gnu/wireshark/extcap/ebpfdump --capture --extcap-interface ebpf --fifo /tmp/wireshark\_extcap\_ebpf\_20191007234339\_IIdYnh --ifname veth24b4614d@kube-dns-6bfbdd666c-5jbmx





## • Start the container (container eth0 172.17.0.2)

\$ docker run -d --name=Jupyter\_Test --rm -p 4443:8888 jupyter/datascience-notebook

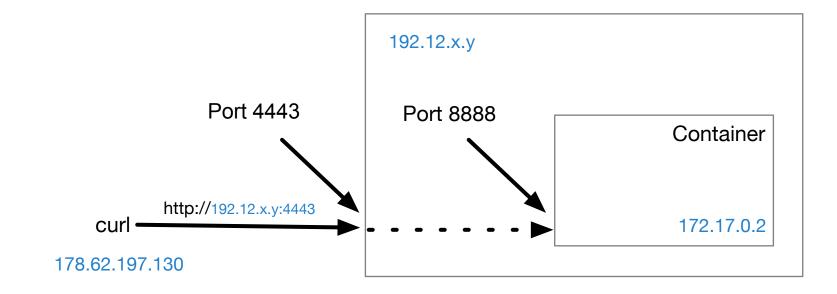
Connect from remote

curl http://host\_ip:4443

1570441451.556130 [eth0][Rcvd][IPv4/TCP][pid/tid: 10713/10713 [/opt/conda/bin/python3.7], uid/ gid: 1000/100][father pid/tid: 10594/0 [/opt/conda/bin/tini], uid/gid: 1000/100][ador: 178.62.197.130:60905 <-> 172.17.0.2:8888][ACCEPT][containe1D: e6296af6\_a71795c60ff6d5034834ed\_216b598658a7111cad42aide9ffe67ee][docker\_name: Jupyter\_Test. Host Container

Remote IP Container IP Mapped Port Local Port







 Linux DNAT (Destination NAT) does the magic mapping ports and IP addresses

# lsof -i -n|grep 4443 docker-pr 16671 root 4u IPv6 124484430 0t0 TCP \*:4443 (LISTEN)

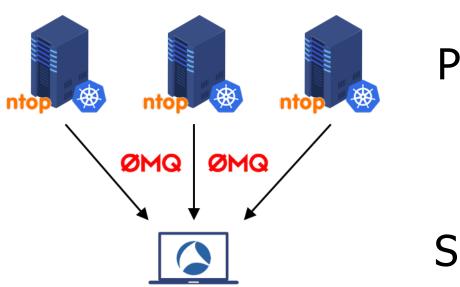
# iptables -L -t nat |grep 4443 DNAT tcp -- anywhere anywhere tcp dpt:4443 to:172.17.0.2:8080

# ps auxw|grep 16671 root 16671 0.0 0.0 378868 2752 ? SI 11:54 0:00 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 4443 -container-ip 172.17.0.2 -container-port 8080 Merging Events with Packets: Headers [4/4]



- As you can see with eBPF we observe
  - Remote IP address and port
  - Container IP and local port
  - No host information reported in events (transparent to the event).
- This means that events can be mapped to packets only on vethX interfaces as on the physical host interface packets will not have the same 5-tuple of the events.





## Publishers

## Subscriber





• Enable flow collection on the host where Wireshark is running (1:N topology)

	🔀 Wireshark · Interface Options: eBPF interface: ebpf				
	eBPF Events				
Interface Name	eBPF Remote Events (ZMQ)				
	Pod kube-dns-6bfbdd666c-5jbmx, Namespace kube-system				
🗸 Save paramete	Pod monitoring-influxdb-grafana-v4-78777c64c8-k8c26, Namespace kube-system				
Restore Default cbr0					
	flannel.1				
	enp0s5				
	veth31d2abd9				
	vethfb526f98				
	lo				
	docker0				



# Remote Flow Export



- Each remote host needs to run
  - ebpflowexport -z "tcp://<wireshark PC>:6789c"
  - Flows are exported and sent in binary format on the "ebpf" topic.
  - The extcap plugin receives the flows and passes them to Wireshark



- Extcap module ported to MacOS (and potentially on other platforms such as Windows)

ZMQ flow collection
allows events to be
delivered remotely

## eBPF on non-Linux OSs



App	ply a display filter <3#/>				Expression .
σ.	Time Source	Destination	Protocol	Info	
	598 56,859385 N/A	N/A	eBPF	UDP Send	
	509 58,288777 N/A	N/A	eBPF	UDP Send	
	518 58,288810 N/A	N/A	OBPE	UDP Send	
	511 58.288820 N/A	N/A	eBPF	UDP_Send	
	512 58.288827 N/A	N/A	eBPF	UDP_Send	
1	513 58.288833 N/A	N/A	eBPF	UDP_Send	
	514 58.288839 N/A	N/A	OBPF	UDP_Send	
	515 58.689572 N/A	N/A	OBPF	UDP_Send	
	516 58.711157 N/A	N/A	eBPF	UDP_Send	
			-		
	Event direction: Sent				
	Event type: UDP_Send				
	IPv4 src address: localhost (127.0.0.1)				
	IPv4 dst address: localhost (127.0.0.1)				
	Event protocol: UDP				
	Event source port: 36267				
	Event destination port: 53				
	Event Process PID: 10707				
	Event Process TID: 10645 Event Process UID: 65534				
	Event Process GID: 65534				
	Event Process Task: sidecar				
	Event Father PID: 18682				
	Event Father TID: 0				
	Event Father UID: 0				
	Event Father GID: 0				
	Event Father Task: containerd-shin				
	Event Container Id: a859ee8bd645074fb9ac68ac87bed	551a566fr4f4a1015645554777af0a99507			
		0			
868					
605					
leð					
fð					
102					
18					
138 148	c2 55 00 80 d0 fc 1a b2 c2 55 00 08 a0 78 1a 1 c2 55 00 80 00 08 08 80 00 80 80 00 80 80 80				



## Future Work: Android

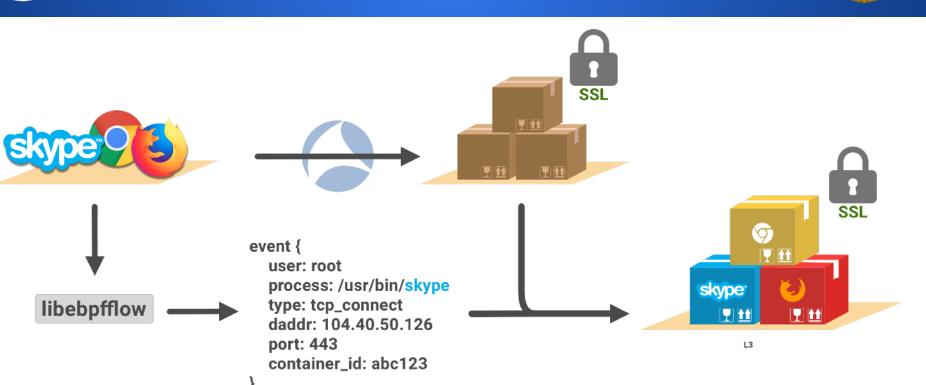


## • eBPF is just being supported on Android...

Set up	Design More 👻	Q Search	LANGUAGE -	GO TO CODE SIGN IN
Design overview ARCHITECTURE	COMPATIBILITY DISPLA	Y SETTINGS TESTS		
Overview Modular System Components Hardware Abstraction Layer (HAL) Kernel Overview Stable Releases & Updates Android Common Kernels Modular Kernels Interface Requirements Extending the kernel with eBPF Ion ABI Changes Configuration Kernel Hardening SquashFS LLDB Debugging Network Tests	Contents V About eBPF Android BPF loader Format of an Android Format of the Android 	the kernel with eB	BPF programs at boot	

### https://source.android.com/devices/architecture/kernel/bpf









# Thank You